Exact algorithms for the repetition-bounded longest common subsequence problem

Exact Algorithms for the Repetition-Bounded Longest Common Subsequence Problem[⋆]

Yuichi Asahiro[a], Jesper Jansson[b], Guohui Lin[c], Eiji Miyano[d,*], Hirotaka Ono[e], Tadatoshi Utashima[d]

[a]*Kyushu Sangyo University, Fukuoka, Japan*
[b]*The Hong Kong Polytechnic University, Kowloon, Hong Kong*
[c]*University of Alberta, Edmonton, Canada*
[d]*Kyushu Institute of Technology, Iizuka, Japan*
[e]*Nagoya University, Nagoya, Japan*

Abstract

In this paper, we study exact, exponential-time algorithms for a variant of the classic Longest Common Subsequence problem called the Repetition-Bounded Longest Common Subsequence problem (or RBLCS, for short): Let an *alphabet* S be a finite set of symbols and an *occurrence constraint* C_{occ} be a function $C_{occ} : S \to \mathbb{N}$, assigning an upper bound on the number of occurrences of each symbol in S. Given two sequences X and Y over the alphabet S and an occurrence constraint C_{occ}, the goal of RBLCS is to find a longest common subsequence of X and Y such that each symbol $s \in S$ appears at most $C_{occ}(s)$ times in the obtained subsequence. The special case where $C_{occ}(s) = 1$ for every symbol $s \in S$ is known as the Repetition-Free Longest Common Subsequence problem (RFLCS) and has been studied previously; e.g., in [1], Adi *et al.* presented a simple (exponential-time) exact algorithm for RFLCS. However, they did not analyze its time complexity in detail, and to the best of our knowledge, there are no previous results on the running times of any exact

[⋆]A preliminary version of the paper appeared in Proceedings of the 13th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2019), Vol. 11949 of Lecture Notes in Computer Science, Springer, pp. 1–12 (2019).

[*]Corresponding author

Email addresses: asahiro@is.kyusan-u.ac.jp (Yuichi Asahiro), jesper.jansson@polyu.edu.hk (Jesper Jansson), guohui@ualberta.ca (Guohui Lin), miyano@ces.kyutech.ac.jp (Eiji Miyano), ono@nagoya-u.jp (Hirotaka Ono), utashima.tadatoshi965@mail.kyutech.jp (Tadatoshi Utashima)

algorithms for this problem. Without loss of generality, we will assume that $|X| \leq |Y|$ and $|X| = n$. In this paper, we first propose a simpler algorithm for RFLCS based on the strategy used in [1] and show explicitly that its running time is $O(1.44225^n)$. Next, we provide a dynamic programming (DP) based algorithm for RBLCS and prove that its running time is $O(1.44225^n)$ for any occurrence constraint C_{occ}, and even less in certain special cases. In particular, for RFLCS, our DP-based algorithm runs in $O(1.41422^n)$ time, which is faster than the previous one. Furthermore, we prove NP-hardness and APX-hardness results for RBLCS on restricted instances.

Keywords: Repetition-bounded longest common subsequence problem, repetition-free, exponential-time exact algorithms, dynamic programming, NP-hardness, APX-hardness

1. Introduction

1.1. Longest Common Subsequence *problems*

An *alphabet* S is a finite set of *symbols*. Let X be a sequence over the alphabet S and $|X|$ be the length of the sequence X. For example, $X = \langle x_1, x_2, \cdots, x_n \rangle$ is a sequence of length n, where $x_i \in S$ for $1 \leq i \leq n$, i.e., $|X| = n$. For a sequence $X = \langle x_1, x_2, \cdots, x_n \rangle$, another sequence $Z = \langle z_1, z_2, \cdots, z_c \rangle$ is a *subsequence* of X if there exists a strictly increasing sequence $\langle i_1, i_2, \cdots, i_c \rangle$ of indices of X such that for all $j = 1, 2, \cdots, c$, we have $x_{i_j} = z_j$. Then, we say that a sequence Z is a *common subsequence* of X and Y if Z is a subsequence of both X and Y. Given two sequences X and Y as input, the goal of the Longest Common Subsequence problem (LCS) is to find a *longest* common subsequence of X and Y.

LCS is a fundamental problem and has a long history [5, 12, 16, 27]. The comparison of sequences via a longest common subsequence has been applied in several contexts where we want to find the maximum number of symbols that appear in the same order in two sequences. LCS is considered to be an important computational primitive in a variety of applications such as bioinformatics [3, 4, 20, 22], data compression [26], spelling correction [21, 27], and file comparison [2] since LCS plays a key role in measuring various types of sequence similarity.

LCS has been deeply investigated, and polynomial-time algorithms are well-known [16, 17, 22, 23, 27]. It is possible to generalize LCS to a set of three or more sequences; the goal is to compute a longest common subsequence of all input sequences. If the number of sequences is part of the input, then LCS of multiple sequences is NP-hard even on binary alphabet [19] and it is not approximable within factor $O(n^{1-\varepsilon})$ on arbitrary alphabet for sequences of length n and any constant $\varepsilon > 0$ [20]. Furthermore, some researchers introduced a constraint on the number of symbol occurrences in the solution. Bonizzoni *et al.* considered the Exemplar Longest Common Subsequence problem (ELCS) [10, 24]. In ELCS, the alphabet S of symbols is divided into the mandatory alphabet S_m and the optional alphabet S_o, and ELCS restricts the numbers of symbol occurrences in S_m and S_o in the obtained solution. ELCS is APX-hard even for instances of two sequences [10]. In [11], Bonizzoni *et al.* proposed the following Doubly-Constrained Longest Common Subsequence problem (DCLCS): Let a sequence constraint C be a set of sequences over an alphabet S and let an occurrence constraint C_{occ} be a function $C_{occ} : S \to \mathbb{N}$, assigning an upper bound on the number of occurrences of each symbol in S. Given two sequences X and Y over the alphabet S, a sequence constraint C, and an occurrence constraint C_{occ}, the goal of DCLCS is to find a longest common subsequence Z of X and Y such that each sequence in C is a subsequence of Z and Z contains at most $C_{occ}(s)$ occurrences of each symbol $s \in S$. Bonizzoni *et al.* showed that DCLCS is NP-hard over an alphabet of three symbols [11].

Adi *et al.* introduced the Repetition-Free Longest Common Subsequence problem (RFLCS) [1]: Given two sequences X and Y over an alphabet S, the goal of RFLCS is to find a "*repetition-free*" longest common subsequence of X and Y, where each symbol appears at most once in the obtained subsequence. In [1], Adi *et al.* proved that RFLCS is APX-hard even if each symbol appears at most twice in each of the given sequences.

$$X = TGACTCTGTGCA$$
$$Y = TGCTCAGTGCAC$$
$$Z = TGCTCGTA$$
$$Z' = TGAC$$

Figure 1: If two sequences X and Y, and an occurrence constraint $C_{occ}(A) = 1$, $C_{occ}(C) = C_{occ}(G) = 2$ and $C_{occ}(T) = 3$ are given as input, then Z is a repetition-bounded longest common subsequence satisfying the occurrence constraint C_{occ}. As another example, if $C_{occ}(A) = C_{occ}(C) = C_{occ}(G) = C_{occ}(T) = 1$, then Z' is a solution (i.e., Z' is *repetition-free*).

1.2. Our new results

In this paper we study exact, exponential-time algorithms for RFLCS and its general form, called the Repetition-Bounded Longest Common Subsequence problem (RBLCS for short): Let $S = \{s_1, s_2, \cdots, s_k\}$ be an alphabet of k symbols. Recall that C_{occ} is an occurrence constraint $C_{occ} : S \to \mathbb{N}$, assigning an upper bound on the number of occurrences of each symbol in S. Given two sequences X and Y over the alphabet S and an occurrence constraint C_{occ}, the goal of RBLCS is to find a "*repetition-bounded*" longest common subsequence of X and Y, where each symbol s_i appears at most $C_{occ}(s_i)$ times in the obtained subsequence for $i = 1, 2, \cdots, k$. See Figure 1 for examples. The special case where $C_{occ}(s_i) = r$ for every $i = 1, 2, \cdots, k$ is referred to as the r-Repetition-Bounded Longest Common Subsequence problem (r-RBLCS for short). Note that the special case 1-RBLCS is identical to RFLCS. Also, it is easy to see that RBLCS is a special case of DCLCS where the sequence constraint C satisfies $C = \emptyset$. In [1], Adi *et al.* presented a simple (exponential-time) exact algorithm for 1-RBLCS, whose basic strategy is to enumerate all the subsequences consisting of representative symbols. However, they did not analyze its time complexity in detail as their focus was on establishing polynomial-time solvability and polynomial-time approximability. To the best of our knowledge, there are no previous results on the running times of any exact algorithms for this problem.

Without loss of generality, we will assume that $|X| \leq |Y|$ and $|X| = n$. The contributions of this paper are summarized as follows:

1. We propose a simple algorithm for RFLCS based on the strategy used in [1] and show explicitly that its running time is $O(1.44225^n)$.
2. We provide a dynamic programming (DP) based algorithm for RBLCS and prove that its running time is $O(1.44225^n)$ for any occurrence constraint C_{occ}, and even less in certain special cases. In particular, for RFLCS, our DP-based algorithm runs in $O(1.41422^n)$ time, which is faster than the previous one.
3. The NP-hardness of RFLCS implies that RBLCS is NP-hard in general. In this paper we prove that for any integer $r \geq 2$, r-RBLCS remains NP-hard even if each symbol appears exactly r or $r + 1$ times in each of the given two sequences. Furthermore, we prove that r-RBLCS is APX-hard if every symbol appears exactly r or $2r$ times in each of the given two sequences.

1.3. Related work

Although this paper focuses on the exact exponential algorithms, we here make a brief survey on previous results for RFLCS, from the viewpoints of heuristic, approximation and parameterized algorithms. In [1], Adi *et al.* introduced first heuristic algorithms for RFLCS. After that, several (meta)heuristic algorithms for RFLCS were proposed in [7, 8, 13, 25]. A detailed comparison of those metaheuristic algorithms was given in [9]. As for the approximability of RFLCS, Adi *et al.* showed [1] that RFLCS admits an $\mathtt{occ}_{max}$-approximation algorithm, where $\mathtt{occ}_{max}$ is the maximum number of occurrences of each symbol in one of the two input sequences. In [6], Blin *et al.* presented a randomized fixed-parameter algorithm for RFLCS parameterized by the size of the solution. In [15], Fernandes and Kiwi studied the asymptotic behavior of the length of a repetition-free longest common subsequence of two *random* sequences such that each symbol appears randomly, uniformly and independently.

1.4. Organization

The rest of the paper is organized as follows: Section 2 introduces notation which will be used throughout the paper, and then gives the formal definition of RBLCS. In Section 3, we present simple exact algorithms based on the strategy used in [1] for RFLCS and r-RBLCS and analyze their running times in detail. Then, we design the $O(1.44225^n)$-time DP-based algorithm for RBLCS in Section 4. Section 5 shows the NP-hardness and the APX-hardness of r-RBLCS on restricted instances. Finally, we conclude the paper in Section 6. The notation used throughout the paper is summarized in the appendix.

2. Preliminaries

Let $S = \{s_1, s_2, \cdots, s_k\}$ be an *alphabet*, i.e., a finite set of k symbols. Let $X = \langle x_1, x_2, \cdots, x_n\rangle$ and $Y = \langle y_1, y_2, \cdots, y_n\rangle$ be the given two sequences as input of RBLCS and Z be a common subsequence of X and Y. For the sequence X, the subsequence $\langle x_i, \cdots, x_j\rangle$ is denoted by $X_{i..j}$. Then, we define the ith *prefix* of X, for $i = 1, \cdots, n$, as $X_{1..i} = \langle x_1, x_2, \cdots, x_i\rangle$. Also, we define the ith *suffix* of X, for $i = 1, \cdots, n$, as $X_{i..n} = \langle x_i, x_{i+1}, \cdots, x_n\rangle$. $X_{1..n}$ is X. Similarly, we define the jth prefix and the jth suffix of Y, for $j = 1, \cdots, m$, as $Y_{1..j} = \langle y_1, y_2, \cdots, y_j\rangle$ and $Y_{j..n} = \langle y_j, y_{j+1}, \cdots, y_m\rangle$, respectively.

Without loss of generality, we assume that both X and Y have all k symbols in S. Let $occ(X, s_i)$, $occ(Y, s_i)$ and $occ(Z, s_i)$ be the numbers of occurrences of s_i in X, Y, and Z, respectively, and thus $occ(X, s_i) \geq 1$ and $occ(Y, s_i) \geq 1$ for every symbol s_i. Let C_{occ} be an occurrence constraint. The Repetition-Bounded Longest Common Subsequence problem (RBLCS) can be formally defined as follows:

Repetition-Bounded Longest Common Subsequence problem (RBLCS)

Input: A pair of sequences X and Y, and an occurrence constraint C_{occ}.

Goal: Find a longest common subsequence Z of X and Y such that $occ(Z, s) \leq C_{occ}(s)$ is satisfied for every $s \in S$.

We call such a sequence Z a *repetition-bounded* longest common subsequence. The special case where $C_{occ}(s_i) = r$ for every $i = 1, 2, \cdots, k$ is referred to as the r-Repetition-Bounded Longest Common Subsequence problem (r-RBLCS). Also, 1-RBLCS is often called the Repetition-Free Longest Common Subsequence problem (RFLCS).

When presenting the time complexity of algorithms, we often round the base of exponential functions up to the fifth digit after the decimal point. That is, for example, the running time $O((\sqrt{2})^n)$ is written as $O(1.41422^n)$ since $\sqrt{2} = 1.414213562...$ and thus $\sqrt{2} < 1.41422$. Furthermore, since $(\sqrt{2})^n poly(n)$ is sandwiched between 1.41421^n and 1.41422^n for every polynomial $poly(n)$ of n and sufficiently large n, we write $O((\sqrt{2})^n poly(n))$ as $O(1.41422^n)$.

3. Warm-up Algorithms

In this section, we first focus on RFLCS, i.e., 1-RBLCS. The following brute-force exact algorithm for RFLCS obviously runs in $O(2^n \cdot n \cdot m)$ time for two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \leq |Y|$: (i) First create all the subsequences of X, denoted by X_1 through X_{2^n}. Then, (ii) obtain a longest common subsequence of X_i and Y for each i $(1 \leq i \leq 2^n)$ by using an $O(|X_i| \cdot m)$-time algorithm for LCS [22, 23, 27]. Finally, (iii) find a repetition-free longest subsequence among those 2^n common subsequences obtained in (ii) and output it.

In [1], Adi *et al.* presented the following algorithm for RFLCS, which is more sophisticated than the naive algorithm above: Let S be an alphabet of symbols. Suppose that each symbol in $S_X \subseteq S$ appears in X fewer times than Y, and $S_X = \{s_1, s_2, \cdots, s_{|S_X|}\}$. Also, let $S_Y = S \setminus S_X$ and $S_Y = \{s_{|S_X|+1}, s_{|S_X|+2}, \cdots, s_{|S|}\}$. (i) The algorithm creates all the subsequences X_1 through X_{N_X} of the input sequence X such that all the symbols in S_X occur *exactly once*, but all the occurrences of symbols in S_Y are kept in X_i for every

$1 \le i \le N_X$. Also, the algorithm creates all the subsequences Y_1 through Y_{N_Y} of the input sequence Y such that all the symbols in S_Y occur *exactly once*, but all the occurrences of symbols in S_X are kept in Y_j for every $1 \le j \le N_Y$. Then, (ii) obtain a longest common subsequence of X_i and Y_j for every pair of i and j ($1 \le i \le N_X$ and $1 \le j \le N_Y$) by using an $O(|X_i| \cdot |Y_j|)$-time algorithm for the original LCS. Finally, (iii) find the longest subsequence among $N_X \cdot N_Y$ common subsequences obtained in (ii), which must be repetition-free, and output it. Clearly, the running time of this method is $O(N_X \cdot N_Y \cdot n \cdot m)$. In [1], Adi *et al.* only claimed that if the number of symbols which appear twice or more in X and Y is bounded above by some constant, say, c, then the running time is $O(m^c \cdot n \cdot m)$, i.e., RFLCS is solvable in polynomial time. However, no upper bound on $N_X \cdot N_Y$ was given in [1].

3.1. Repetition-Free LCS

In this subsection we consider an algorithm called ALG, based on the same strategy as the one in [1] for RFLCS: (i) First create all the subsequences X_1 through X_N of the input sequence X such that every symbol appears *exactly once* in X_i for $1 \le i \le N$ in $O(N \cdot n)$ time. Then, (ii) obtain a longest common subsequence of X_i and Y in $O(n \cdot m)$ time for each i ($1 \le i \le N$). Finally, (iii) find a repetition-free longest subsequence among N common subsequences obtained in (ii) and output it. Therefore, the running time of ALG is $O(N \cdot n \cdot m)$. It is important to note that ALG is identical to Adi *et al.*'s algorithm in [1] if $S_X = S$ and thus $S_Y = \emptyset$.

A very simple argument gives us the first upper bound on N and the running time of ALG:

Theorem 1. *The running time of ALG is $O(1.44467^n)$ for RFLCS on two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \le |Y|$.*

Proof. Recall that X has k symbols, s_1 through s_k, and s_i occurs $occ(X, s_i)$ times in X for each integer i, $1 \le i \le k$. Since the number N of subsequences in X created in (i) of ALG is bounded by the number of combinations of k symbols, the following is satisfied:

$$N \leq \prod_{i=1}^{k} occ(X, s_i).$$

From the inequality of arithmetic and geometric means, we have:

$$N \leq \left((\sum_{i=1}^{k} occ(X, s_i))/k \right)^k \leq (n/k)^k.$$

Here, by setting $p \overset{\text{def}}{=} n/k \in \mathbb{R}^+$, we have:

$$N \leq (p)^{n/p} = (p^{1/p})^n.$$

Note that the value of $p^{1/p}$ becomes the maximum when $p = e$, where e denotes Euler's number. That is, N is bounded above by $e^{n/e} < 1.444668^n$. Therefore, the running time of `ALG` is $O(1.444668^n \cdot n \cdot m) = O(1.44467^n)$. □

A more refined estimate yields a smaller upper bound on N, which gives us the improved running time of `ALG`:

Theorem 2. *The running time of `ALG` is $O(1.44225^n)$ for RFLCS on two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \leq |Y|$.*

Proof. Let $\max_{1 \leq i \leq k} \{occ(X, s_i)\} = \mathtt{occ}_{max}$. Also, let $S_i = \{s_j \mid occ(X, s_j) = i\}$ for $1 \leq i \leq \mathtt{occ}_{max}$. That is, S_i is a set of symbols which appear exactly i times in X. Let $n_i = i \times |S_i|$. Since each symbol in S_i appears i times in X, the following equality holds:

$$\sum_{i=1}^{\mathtt{occ}_{max}} n_i = n. \tag{1}$$

In the following, we show a smaller upper bound on N than that in the proof of Theorem 1. From the fact that $n_i = i \times |S_i|$, one sees that the following equality holds:

$$N \leq \prod_{i=1}^{k} occ(X, s_i) = \prod_{i=1}^{\mathtt{occ}_{max}} i^{n_i/i}. \tag{2}$$

Here, from the inequality of arithmetic and geometric means, the following is obtained:

$$\left(\prod_{i=1}^{\mathsf{occ}_{max}} \left(i^{1/i}\right)^{n_i}\right)^{1/\sum_{i=1}^{\mathsf{occ}_{max}} n_i} \le \frac{\sum_{i=1}^{\mathsf{occ}_{max}} \left(i^{1/i}\right) \cdot n_i}{\sum_{i=1}^{\mathsf{occ}_{max}} n_i}. \tag{3}$$

From the equations (1), (2), and (3), we get:

$$N \le \left(\frac{\sum_{i=1}^{\mathsf{occ}_{max}} \left(i^{1/i}\right) \cdot n_i}{n}\right)^n. \tag{4}$$

Now, it is important to note that $i \in \mathbb{N}$, i.e., i is a positive integer while $p = n/k$ is a positive real in the proof of the previous theorem. Therefore, by a simple calculation, one can verify that the following is true:

$$\max_{i \in \mathbb{N}} \left\{i^{1/i}\right\} = 3^{1/3}. \tag{5}$$

Hence, we can bound the number N of all the possible repetition-free common subsequences as follows:

$$\begin{aligned} N &\le \left(\frac{\sum_{i=1}^{\mathsf{occ}_{max}} 3^{1/3} \cdot n_i}{n}\right)^n \\ &= \left(\frac{3^{1/3} \cdot \sum_{i=1}^{\mathsf{occ}_{max}} n_i}{n}\right)^n \\ &= \left(3^{1/3}\right)^n \\ &< 1.4422496^n. \end{aligned}$$

As a result, the running time of our algorithm is $O(1.4422496^n \cdot n \cdot m) = O(1.44225^n)$. This completes the proof. □

Corollary 1. *There is an $O(\mathsf{occ}^{n/\mathsf{occ}} \cdot n \cdot m)$-time algorithm to solve RFLCS*
190 *for two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \le |Y|$ when every symbol occurs in X exactly occ times.*

Proof. By the assumption, $\mathsf{occ} \times |S_{\mathsf{occ}}| = n$ and thus $|S_{(occ)'}| = 0$ for $\mathsf{occ} \ne \mathsf{occ}'$. From the inequality (4), one can easily obtain the following:

$$N \le \left(\mathsf{occ}^{1/\mathsf{occ}}\right)^n.$$

□

Table 1 shows the running time T for each $\mathsf{occ} = 2, 3, \cdots, 8$:

Table 1: Occurrence occ and running time T

occ	2	3	4	5	6	7	8
T	1.41422^n	1.44225^n	1.41422^n	1.37973^n	1.34801^n	1.32047^n	1.29684^n

3.2. r-Repetition-Bounded LCS, $r \geq 2$

In this subsection we consider exact exponential algorithms for r-RBLCS. First, by a straightforward extension of the algorithm for RFLCS, we can design the following algorithm for r-RBLCS, named $\mathtt{ALG}_r$: First, (i) create all the subsequences X_1 through X_N of the input sequence X such that each symbol s appears *exactly* r times in X_i for $1 \leq i \leq N$ if X has more than r s's; otherwise, all the occurrences of s in X are included in X_i. Each subsequence X_i can be created in $O(n)$ time and thus the total running time of (i) is $O(N \cdot n)$. Then, (ii) obtain a longest common subsequence of X_i and Y in $O(n \cdot m)$ time for each i ($1 \leq i \leq N$). Finally, (iii) find a longest subsequence among N common subsequences obtained in (ii), which has at most r occurrences of every symbol, and output it. Therefore, the running time is $O(N \cdot n \cdot m)$.

Again, suppose that X has k symbols, $s_1, s_2, \cdots, s_k$, and s_i occurs $occ(X, s_i)$ times in X for each integer i, $1 \leq i \leq k$, and $\max_{1 \leq i \leq k} \{occ(X, s_i)\} = \mathsf{occ}_{max}$. Let $S_i = \{s_j \mid occ(X, s_j) = i\}$ for $1 \leq i \leq \mathsf{occ}_{max}$ and $n_i = i \times |S_i|$. Then, we estimate an upper bound on N for each r:

Theorem 3. *For r-RBLCS on two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \leq |Y|$, the running time of $\mathtt{ALG}_r$ is as follows:*

$$O\left(\left(\max_{i \in \mathbb{N}} \left\{ \left(\frac{i - \frac{r-1}{2}}{(r!)^{1/r}} \right)^{r/i} \right\} \right)^n \times n \cdot m \right).$$

Proof. First, the total number N of sequences created in (i) of $\mathtt{ALG}_r$ can be expressed as follows:

$$N = \prod_{i=1}^{k} \binom{\mathsf{occ}_i}{r} = \prod_{i=r+1}^{\mathsf{occ}_{max}} \binom{i}{r}^{n_i/i}.$$

From the inequality of arithmetic and geometric means, we can obtain the following inequality:

$$(i(i-1)(i-2)\cdots(i-r+1))^{1/r} \le \frac{(2i-r+1)r/2}{r} = i - \frac{r-1}{2}.$$

Therefore, N is bounded:

$$\begin{aligned}\prod_{i=r+1}^{\mathsf{occ}_{max}} \binom{i}{r}^{n_i/i} &\le \prod_{i=r+1}^{\mathsf{occ}_{max}} \left(\frac{(i-\frac{r-1}{2})^r}{r!}\right)^{n_i/i} \\ &= \prod_{i=r+1}^{\mathsf{occ}_{max}} \left(\left(\frac{i-\frac{r-1}{2}}{(r!)^{1/r}}\right)^{r/i}\right)^{n_i} \\ &\le \left(\max_{i\in\mathbb{N}}\left\{\left(\frac{i-\frac{r-1}{2}}{(r!)^{1/r}}\right)^{r/i}\right\}\right)^{n}.\end{aligned}$$

This completes the proof. □

We have obtained the specific values of $\max_{i\in\mathbb{N}}\left\{\left(\frac{i-\frac{r-1}{2}}{(r!)^{1/r}}\right)^{r/i}\right\}$, denoted by $N(r)$, and i for r-RBLCS by its empirical implementation. Table 2 shows $N(r)$ and i for each $r = 2, 3, \cdots, 10$.

Table 2: $N(r)$ and i for each r

r	2	3	4	5	6	7	8	9	10
$N(r)$	1.58884	1.66852	1.72013	1.75684	1.78453	1.80630	1.82394	1.83856	1.85091
i	5	7	9	11	13	15	17	19	21

4. Dynamic Programming Algorithms for RBLCS

In this section we design a DP-based algorithm named DP for RBLCS.

4.1. Original LCS

In this subsection, we briefly review the dynamic programming paradigm for the original LCS. For more details, e.g., see [14]. Let $Z_{1..h} = \langle z_1, z_2, \cdots, z_h\rangle$ be any longest common subsequence of the ith prefix $X_{1..i}$ of X and the jth prefix $Y_{1..j}$ of Y. It is well known that LCS has the following optimal-substructure property: (1) If $x_i = y_j$, then $z_h = x_i = y_j$ and $Z_{1..h-1}$ is a longest common subsequence of $X_{1..i-1}$ and $Y_{1..j-1}$. (2) If $x_i \neq y_j$, then (a) $z_h \neq x_i$ implies that $Z_{1..h}$ is a longest common subsequence of $X_{1..i-1}$ and $Y_{1..j}$; (b) $z_h \neq y_j$, then $Z_{1..h}$ is a longest common subsequence of $X_{1..i}$ and $Y_{1..j-1}$.

We define $L(i,j)$ to be the length of a longest common subsequence of $X_{1..i}$ and $Y_{1..j}$. Then, the above optimal substructure of LCS gives the following recursive formula:

$$L(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ L(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max\{L(i, j-1), L(i-1, j)\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

The DP algorithm for the original LCS computes each value of $L(i,j)$ and stores it into a two-dimensional table L of size $(n+1) \times (m+1)$ in row-major order.

In the case of RBLCS, we have to count the number of occurrences of every symbol in the prefix of Z. In the following we show a modified recursive formula and a DP-based algorithm for RBLCS.

4.2. Repetition-Bounded LCS

A trivial implementation of a dynamic programming approach might be to use the DP-based algorithm for LCS for *multiple* sequences: For RFLCS, we first generate all the permutations of k symbols, i.e., $k!$ repetition-free sequences of k symbols, say, X_1 through $X_{k!}$ and then obtain a longest common subsequence of X_i, X, and Y for each i $(1 \leq i \leq k!)$ by using an $O(|X_i| \cdot n \cdot m)$-time DP-based algorithm solving LCS for multiple (three) sequences proposed in [18]. Therefore, the total running time is $O(k! \cdot k \cdot n \cdot m)$. For RBLCS, we first generate all the permutation of $\sum_{i=1}^{k} C_{occ}(s_i)$ multiple symbols and then obtain a longest common subsequence Z such that $occ(Z, s_i) \leq C_{occ}(s_i)$ is satisfied for every $s_i \in S$. Let $N = \sum_{i=1}^{k} C_{occ}(s_i)$. Then, the running time is $O(N! \cdot N \cdot n \cdot m)$, which is polynomial if N is constant.

In the following we design a faster DP-based algorithm $\mathtt{DP}$. Let $S_{>C_{occ}} = \{s_i \mid occ(X, s_i) > C_{occ}(s_i)\}$. Now suppose that $|S_{>C_{occ}}| = \ell$ and, without loss of generality, $S_{>C_{occ}} = \{s_1, s_2, \cdots, s_\ell\}$. Then, we prepare an "occurrence" vector of length ℓ, denoted by $\boldsymbol{v} = (v_1, v_2, \cdots, v_\ell)$, where the pth component v_p corresponds to the pth symbol s_p for $1 \le p \le \ell$ and $v_p \in \{0, 1, \cdots, C_{occ}(s_p)\}$. Roughly speaking, DP uses the occurrence vector $\boldsymbol{v}$ as an upper bound of occurrences of every symbol in an intermediate solution, in order not to break the occurrence constraint; it can therefore compute a subproblem of finding a repetition-bounded longest common subsequence of $X_{1..i}$ and $Y_{1..j}$. Note that the number of possibilities in the occurrence vector is $\prod_{i=1}^{\ell}(C_{occ}(s_i) + 1)$.

For the occurrence vector $\boldsymbol{v} = (v_1, v_2, \cdots, v_{p-1}, v_p, v_{p+1} \cdots, v_\ell)$, we define a new vector $\boldsymbol{v}|_{p=q} = (v_1, v_2, \cdots, v_{p-1}, q, v_{p+1}, \cdots, v_\ell)$. Note that if $v_p = q$ in $\boldsymbol{v}$, then $\boldsymbol{v}|_{p=q} = \boldsymbol{v}$. Let $\mathbf{0}$ be an ℓ-dimensional 0-vector, i.e., the length of $\mathbf{0}$ is ℓ and all ℓ components are 0. Also, let $\boldsymbol{C_{occ}}$ be an ℓ-dimensional vector such that the length of $\boldsymbol{C_{occ}}$ is ℓ and the pth component is $C_{occ}(s_p)$ for $1 \le p \le \ell$.

Similarly to the previous subsection, we define $L(i, j, \boldsymbol{v})$ to be the length of a repetition-bounded longest common subsequence of $X_{1..i}$ and $Y_{1..j}$ satisfying the occurrence vector $\boldsymbol{v}$, i.e., the length of the subsequence which does not break the occurrence constraint. Our algorithm for RBLCS computes each value of $L(i, j, \boldsymbol{v})$ and stores it into a three-dimensional table L of size $(n + 1) \times (m + 1) \times \prod_{i=1}^{\ell}(C_{occ}(s_i) + 1)$.

Theorem 4 (Optimal substructure of RBLCS)**.** *Consider the ith prefix $X_{1..i}$ of X and the jth prefix $Y_{1..j}$ of Y. Suppose that $S_{>C_{occ}} = \{s_1, s_2, \cdots, s_\ell\}$ be a set of ℓ symbols such that each s_i occurs at least $C_{occ}(s_i) + 1$ times in X. Let $Z_{1..h} = \langle z_1, z_2, \cdots, z_h\rangle$ be any repetition-bounded longest common subsequence of $X_{1..i}$ and $Y_{1..j}$ satisfying an occurrence vector $\boldsymbol{v}$. Then, the followings are satisfied:*

(1) *If $x_i = y_j = s_p$ and $s_p \notin S_{>C_{occ}}$, then $z_h = s_p$ and $Z_{1..h-1}$ is a repetition-bounded longest common subsequence of $X_{1..i-1}$ and $Y_{1..j-1}$ satisfying $\boldsymbol{v}$.*

(2) *If $x_i = y_j = s_p$, $s_p \in S_{>C_{occ}}$ and $v_p > 0$, then $z_h = s_p$ implies that $Z_{1..h-1}$ is a repetition-bounded longest common subsequence of $X_{1..i-1}$ and $Y_{1..j-1}$ satisfying $\boldsymbol{v}|_{p=v_p-1}$.*

(3) *If $x_i = y_j = s_p$, $s_p \in S_{>C_{occ}}$ and $v_p = 0$, then $z_h \neq s_p$ and $Z_{1..h}$ is a repetition-bounded longest common subsequence of $X_{1..i-1}$ and $Y_{1..j-1}$ satisfying $\boldsymbol{v}$.*

(4) *If $x_i \neq y_j$, then*

(a) *$z_h \neq x_i$ implies that $Z_{1..h}$ is a repetition-bounded longest common subsequence of $X_{1..i-1}$ and $Y_{1..j}$ satisfying $\boldsymbol{v}$;*

(b) *$z_h \neq y_j$ implies that $Z_{1..h}$ is a repetition-bounded longest common subsequence of $X_{1..i}$ and $Y_{1..j-1}$ satisfying $\boldsymbol{v}$.*

Proof. We will verify (1) through (4):

(1) If $z_h \neq x_i$, then by appending $x_i = y_j = s_p$ to $Z_{1..h}$, we can obtain a repetition-bounded common subsequence of $X_{1..i}$ and $Y_{1..j}$ of length $h+1$ satisfying $\boldsymbol{v}$ since the number of s_p's in Z is at most $C_{occ}(s_p) - 1$ from the condition $s_p \notin S_{>C_{occ}}$. This contradicts the assumption that $Z_{1..h}$ is a repetition-bounded longest common subsequence of $X_{1..i}$ and $Y_{1..j}$ satisfying $\boldsymbol{v}$. Therefore, $z_h = x_i = y_j$ holds. What we have to do is to prove that the prefix $Z_{1..h-1}$ is a repetition-bounded longest common subsequence of $X_{1..i-1}$ and $Y_{1..j-1}$ with length $h-1$ satisfying $\boldsymbol{v}$. For the purpose of obtaining a contradiction, suppose that there exists a repetition-bounded common subsequence Z' of $X_{1..i-1}$ and $Y_{1..j-1}$ with length greater $h-1$ satisfying $\boldsymbol{v}$. Then, by appending $x_i = y_j = s_p$, we obtain a repetition-bounded common subsequence of $X_{1..i}$ and $Y_{1..j}$ whose length is greater than h satisfying $\boldsymbol{v}$, which is a contradiction.

(2) If $z_h = x_i = y_j = s_p$, then $Z_{1..h-1}$ is a repetition-bounded common subsequence of $X_{1..i-1}$ and $Y_{1..j-1}$ such that s_p appears at most $v_p - 1$ times in $Z_{1..h-1}$. Suppose that there exists a repetition-bounded common subsequence Z' of $X_{1..i-1}$ and $Y_{1..j-1}$ with length greater than $h-1$ satisfying $\boldsymbol{v}|_{p=v_p-1}$. Since the pth component of $\boldsymbol{v}|_{p=v_p-1}$ is $v_p - 1$, by appending $x_i = y_j = s_p$ to Z', we obtain a repetition-bounded common subsequence of $X_{1..i}$ and $Y_{1..j}$ whose length is greater than h satisfying $\boldsymbol{v}$, which contradicts the assumption that $Z_{1..h}$ is a repetition-bounded longest common subsequence of $X_{1..i}$ and $Y_{1..j}$ satisfying $\boldsymbol{v}$.

(3) Suppose that there exists a repetition-bounded common subsequence Z' of $X_{1..i-1}$ and $Y_{1..j-1}$ with length greater than h satisfying $\boldsymbol{v}$. From $z_h \neq s_p$, Z' is also a repetition-bounded common subsequence of $X_{1..i}$ and $Y_{1..j}$ satisfying $\boldsymbol{v}$, which again contradicts the assumption that $Z_{1..h}$ is a repetition-bounded longest common subsequence of $X_{1..i}$ and $Y_{1..j}$ satisfying $\boldsymbol{v}$.

(4)(a) ((b), resp.) If there is a repetition-bounded common subsequence Z' of $X_{1..i-1}$ and $Y_{1..j}$ ($X_{1..i}$ and $Y_{1..j-1}$, resp.) with length greater than h satisfying $\boldsymbol{v}$, then Z' would also be a repetition-bounded common subsequence of $X_{1..i}$ and $Y_{1..j}$ satisfying $\boldsymbol{v}$, contradicting the assumption that Z is a repetition-bounded longest common subsequence of $X_{1..i}$ and $Y_{1..j}$ satisfying $\boldsymbol{v}$. □

Then, we can obtain the following recursive formula:

$$L(i,j,\boldsymbol{v}) = \begin{cases} 0 \quad \text{if } i=0 \text{ or } j=0, \\ L(i-1,j-1,\boldsymbol{v})+1 \\ \quad \text{if } i,j>0,\ x_i=y_j=s_p, \text{ and } s_p \notin S_{>C_{occ}} \quad \text{(Case (1))}, \\ L(i-1,j-1,\boldsymbol{v}|_{p=v_p-1})+1 \\ \quad \text{if } i,j>0,\ x_i=y_j=s_p,\ s_p \in S_{>C_{occ}}, \text{ and } v_p>0 \quad \text{(Case (2))}, \\ L(i-1,j-1,\boldsymbol{v}) \\ \quad \text{if } i,j>0,\ x_i=y_j=s_p,\ s_p \in S_{>C_{occ}}, \text{ and } v_p=0 \quad \text{(Case (3))}, \\ \max\{L(i-1,j,\boldsymbol{v}), L(i,j-1,\boldsymbol{v})\} \\ \quad \text{otherwise} \quad \text{(Case (4))}. \end{cases}$$

Here is an outline of our algorithm DP, which computes each value of $L(i,j,\boldsymbol{v})$ and stores it into a three-dimensional table L of size $(n+1)\times(m+1)\times\prod_{i=1}^{\ell}(C_{occ}(s_i)+1)$: Initially, we set $L(i,j,\boldsymbol{v}) = 0$ and $pre(i,j,\boldsymbol{v}) =$ `null` for every i, j, and $\boldsymbol{v}$. Then, the algorithm DP fills entries from $L(1,1,\boldsymbol{0})$ to $L(1,1,\boldsymbol{C_{occ}})$, then from $L(1,2,\boldsymbol{0})$ to $L(1,2,\boldsymbol{C_{occ}})$, next from $L(1,3,\boldsymbol{0})$ to $L(1,3,\boldsymbol{C_{occ}})$, etc. After filling all the entries in the first "two-dimensional plane" $L(1,j,\boldsymbol{v})$, the algorithm fills all the entries in the second two-dimensional plane

$L(2, j, \boldsymbol{v})$, and so on. Finally, DP fills all the entries in the nth plane. The algorithm DP also maintains a three dimensional table pre of size $(n+1) \times (m+1) \times \prod_{i=1}^{\ell}(C_{occ}(s_i)+1)$ to help us construct an optimal repetition-bounded longest subsequence. The entry $pre(i, j, \boldsymbol{v})$ points to the table entry corresponding to the optimal subproblem solution chosen when computing $L(i, j, \boldsymbol{v})$.

Algorithm DP:

Input: Two sequences $X = \langle x_1, x_2, \cdots, x_n \rangle$ and $Y = \langle y_1, y_2, \cdots, y_m \rangle$, and an occurrence constraint C_{occ}.

Output: Repetition-bounded longest common subsequence Z of X and Y.

Initialization: Find $S_{>C_{occ}} = \{s_1, s_2, \cdots, s_k\}$ and then set $L(i, j, \boldsymbol{v}) = 0$ and $pre(i, j, \boldsymbol{v}) =$ null for each i, j, and $\boldsymbol{v}$.

```
1.  for $i = 1$ to $n$
2.      for $j = 1$ to $m$
3.          for $\boldsymbol{v} = \boldsymbol{0}$ to $\boldsymbol{C_{occ}}$
4.              /* Case (1) */
5.              if $x_i == y_j == s_p$ and $s_p \notin S_{>C_{occ}}$
6.                  $L(i, j, \boldsymbol{v}) = L(i-1, j-1, \boldsymbol{v}) + 1$
7.                  $pre(i, j, \boldsymbol{v}) =$ “$(i-1, j-1, \boldsymbol{v})$”
8.              /* Case (2) */
9.              elseif $x_i == y_j == s_p$, $s_p \in S_{>C_{occ}}$, and $v_p > 0$
10.                 $L(i, j, \boldsymbol{v}) = L(i-1, j-1, \boldsymbol{v}|_{p=v_p-1}) + 1$
11.                 $pre(i, j, \boldsymbol{v}) =$ “$(i-1, j-1, \boldsymbol{v}|_{p=v_p-1})$”
12.             /* Case (3) */
13.             elseif $x_i == y_j == s_p$, $s_p \in S_{>C_{occ}}$ and $v_p = 0$
14.                 $L(i, j, \boldsymbol{v}) = L(i-1, j-1, \boldsymbol{v})$
15.                 $pre(i, j, \boldsymbol{v}) =$ “$(i-1, j-1, \boldsymbol{v})$”
16.             /* Case (4)(a) */
17.             elseif $L(i-1, j, \boldsymbol{v}) \geq L(i, j-1, \boldsymbol{v})$
```

```
18.               $L(i, j, \boldsymbol{v}) = L(i-1, j, \boldsymbol{v})$
19.               $pre(i, j, \boldsymbol{v})$ = "$(i-1, j, \boldsymbol{v})$"
20.             /* Case (4)(b) */
21.             else $L(i, j, \boldsymbol{v}) = L(i, j-1, \boldsymbol{v})$
22.                 $pre(i, j, \boldsymbol{v})$ = "$(i, j-1, \boldsymbol{v})$"
```

Termination: Construct a repetition-bounded longest common subsequence Z based on two tables L and pre, and then output Z.

We bound the running time of DP:

Theorem 5. *The running time of DP is $O(\prod_{i=1}^{\ell}(C_{occ}(s_i)+1)\cdot n\cdot m)$ for RBLCS on two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \leq |Y|$.*

Proof. The algorithm DP for RBLCS computes each value of $L(i, j, \boldsymbol{v})$ and stores it into the three-dimensional table L of size $(n+1)\times(m+1)\times\prod_{i=1}^{\ell}(C_{occ}(s_i)+1)$. Clearly, each table entry takes $O(1)$ time to compute. As a result, the running time of DP is $O(\prod_{i=1}^{\ell}(C_{occ}(s_i)+1)\cdot n\cdot m)$. □

By showing that $\prod_{i=1}^{\ell}(C_{occ}(s_i)+1) \leq \left(3^{1/3}\right)^n$ is satisfied, we obtain the following corollary:

Corollary 2. *The running time of DP is $O(1.44225^n)$ for RBLCS on two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \leq |Y|$.*

Proof. Let $|S_{>C_{occ}}| = \ell$ again. Recall that for every $s_i \in S_{>C_{occ}}$, $occ(X, s_i) > C_{occ}(s_i)$. That is, $occ(X, s_i) \geq C_{occ}(s_i) + 1$ since both $occ(X, s_i)$ and $C_{occ}(s_i)$ are integers. Therefore, the following is satisfied:

$$\sum_{i=1}^{\ell}(C_{occ}(s_i)+1) \leq \sum_{i=1}^{\ell} occ(X, s_i) \leq n. \tag{6}$$

Let $C_{\max} = \max_{s_i \in S_{>C_{occ}}}\{C_{occ}(s_i)\}$ be the maximum of the occurrence constraint. Also, let $u_i = |\{s_j \mid occ(X, s_j) = i\}|$ be the number of symbols which appear exactly i times in X for $1 \leq i \leq C_{max}$. One sees that the term $\prod_{i=1}^{\ell}(C_{occ}(s_i)+1)$ in the running time of DP can be rewritten as follows:

$$\prod_{i=1}^{\ell}(C_{occ}(s_i)+1) = \prod_{i=2}^{C_{\max}+1}((i-1)+1)^{u_{i-1}} = \prod_{i=2}^{C_{\max}+1} i^{u_{i-1}} = \prod_{i=2}^{C_{\max}+1} i^{\frac{V_i}{i}}, \quad (7)$$

where the rightmost equality holds by setting $V_i \overset{\text{def}}{=} i \times u_{i-1}$. Then, we can show the following upper bound of the summation from V_2 to $V_{C_{max}+1}$ from the above inequality (6):

$$\sum_{i=2}^{C_{\max}+1} V_i = \sum_{i=1}^{\ell}(C_{occ}(s_i)+1) \leq n. \quad (8)$$

By combining the (in)equalities (5), (7), and (8), we can obtain the following upper bound on $\prod_{i=1}^{\ell}(C_{occ}(s_i)+1)$:

$$\prod_{i=1}^{\ell}(C_{occ}(s_i)+1) \ \leq \ \prod_{i=2}^{C_{\max}+1} 3^{\frac{V_i}{3}} = \left(3^{1/3}\right)^{\sum_{i=2}^{C_{\max}+1} V_i} \leq \left(3^{1/3}\right)^{n}.$$

Therefore, the running time of DP is $O(1.44225^n)$ for RBLCS. □

The algorithm DP works a little bit faster for RFLCS:

Corollary 3. *The running time of DP is $O(1.41422^n)$ for RFLCS on two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \leq |Y|$.*

Proof. It is enough to prepare the three-dimensional table L of size $(n+1) \times (m+1) \times 2^{\ell}$ for RFLCS. Clearly, each table entry takes $O(1)$ time to compute. As a result, the running time of DP is $O(2^{\ell} \cdot n \cdot m)$. Recall that the number $|S_{>C_{occ}}|$ of symbols which appear at least twice in X is defined to be ℓ. This implies that $\ell \leq \frac{n}{2}$. Therefore, $2^{\ell} \leq 2^{n/2} < 1.414214^n$ is satisfied; the running time is $O(1.41422^n)$ for RFLCS. □

The running time for r-RBLCS is as follows:

Corollary 4. *The running time of DP is $O((r+1)^{n/(r+1)} \cdot n \cdot m)$ for r-RBLCS on two sequences X and Y where $|X| = n$, $|Y| = m$, and $|X| \leq |Y|$.*

Proof. We prepare a three-dimensional table L of size $(n+1)\times(m+1)\times(r+1)^{\ell}$ and each table entry takes $O(1)$ time to compute. Clearly $\ell \leq \frac{n}{r+1}$, i.e., $(r+1)^{\ell} \leq (r+1)^{n/(r+1)}$ holds. $\square$

Table 3 shows the running time T of DP for r-RBLCS, $r = 2, 3, \cdots, 8$:

Table 3: Running time T of DP for r-RBLCS

r	2	3	4	5	6	7	8
T	1.44225^n	1.41422^n	1.37973^n	1.34801^n	1.32047^n	1.29684^n	1.27652^n

5. Hardness of RBLCS

The NP-hardness (or the APX-hardness) of RFLCS implies that RBLCS on general instances is also NP-hard. In this section, we investigate the computational complexity of RBLCS on restricted instances. First, we consider r-RBLCS where the instance is a pair of sequences X and Y such that each symbol appears exactly r or $r+1$ times for any integer $r \geq 2$. We can show the following hardness result:

Theorem 6. *For any integer $r \geq 2$, r-RBLCS is NP-hard even if $occ(X, s_i) \in \{r, r+1\}$ and $occ(Y, s_i) \in \{r, r+1\}$ hold for every symbol $s_i \in S$.*

Proof. We prove that the NP-hardness of r-RBLCS by providing the polynomial-time reduction from RFLCS to r-RBLCS. Suppose that a pair of sequences $X = \langle x_1, x_2, \cdots, x_n\rangle$ and $Y = \langle y_1, y_2, \cdots, y_m\rangle$ is an instance of RFLCS such that every symbol appears at most twice in each of two sequences. Recall that RFLCS is NP-hard even if each symbol appears at most twice in each of the given two sequences [1]. Let $S = \{s_1, s_2, \cdots, s_k\}$. Then, we construct the following pair of sequences X^r and Y^r as an instance of r-RBLCS:

$$X^r = \langle x_1, x_2, \cdots, x_n, \overbrace{s_1, \cdots, s_1}^{r-1}, \overbrace{s_2, \cdots, s_2}^{r-1}, \cdots, \overbrace{s_k, \cdots, s_k}^{r-1}\rangle$$

$$Y^r = \langle y_1, y_2, \cdots, y_m, \overbrace{s_1, \cdots, s_1}^{r-1}, \overbrace{s_2, \cdots, s_2}^{r-1}, \cdots, \overbrace{s_l, \cdots, s_k}^{r-1}\rangle$$

That is, the nth prefix of $X^r_{1..n}$ (mth prefix of $Y^r_{1..m}$, resp.) is X (Y, resp.), the next $r-1$ symbols of X^r (Y^r, resp.) are $r-1$ duplicates of s_1, the next $r-1$ symbols of X^r (Y^r, resp.) are $r-1$ duplicates of s_2, etc. This completes the reduction, which can be clearly done in polynomial time. One sees that every symbol appears exactly r or $r+1$ times in each of the two sequences X^r and Y^r.

In the following, we show that there is a repetition-free common subsequence Z of X and Y of length at least c if and only if there is a common subsequence Z^r of X^r and Y^r such that the length $|Z^r|$ is at least $c+k(r-1)$ under the constraint $occ(Z^r, s_i) \leq r$ for every symbol $s_i \in S$.

(Only-if part) Suppose that $Z = \langle z_1, z_2, \cdots, z_c \rangle$ is an optimal solution for RFLCS when its instance is the pair of sequences X and Y. Clearly,

$$Z^r = \langle z_1, z_2, \cdots, z_c, \overbrace{s_1, \cdots, s_1}^{r-1}, \overbrace{s_2, \cdots, s_2}^{r-1}, \cdots, \overbrace{s_k, \cdots, s_k}^{r-1} \rangle$$

is a common subsequence of X^r and Y^r such that $occ(Z^r, s_i) \leq r$ since the cth prefix of Z^r is repetition-free. The length of Z^r is (at least) $c+k(r-1)$.

(If part) Suppose that Z^* is a repetition-bounded longest common subsequence such that $occ(Z^*, s_i) \leq r$ for every symbol $s_i \in S$ and the length of Z^* is at least $c+k(r-1)$. If the number of symbols whose r occurrences are included in Z^* is at most $c-1$, then the length of Z^* must be less than $c+k(r-1)$ by the following calculation (since the remaining $(k-c+1)$ symbols appear at most $r-1$ times):

$$r(c-1) + (r-1)(k-c+1) = c + k(r-1) - 1 < c + k(r-1).$$

That is, there are at least c symbols whose r occurrences are included in Z^*. Suppose that s^*_1 through s^*_c appear r times in Z^*. Observe that each of the $(n+1)$st suffix $X^r_{n+1..n+k(r-1)}$ of X^r and the $(m+1)$st suffix $Y^r_{m+1..m+k(r-1)}$ has exactly $r-1$ occurrences of every symbol $s_i \in S$. This implies that the nth prefix $X^r_{1..n}$ of X^r and the mth prefix $Y^r_{1..m}$ of Y^r has one or two s^*_i's for $i = 1, 2, \cdots, c$. Suppose, for example, that $c = 5$, and $X^r_{1..n}$ and $Y^r_{1..m}$ have the following structure:

$$X^r_{1..n} = \langle \cdots, s_1^*, \cdots\cdots, s_2^*, \cdots, s_1^*, \cdots, s_3^*, s_4^*, \cdots, s_2^*, \cdots, s_5^* \rangle$$
$$Y^r_{1..m} = \langle \cdots, s_1^*, \cdots, s_2^*, s_1^*, \cdots\cdots, s_3^*, \cdots\cdots, s_4^*, \cdots\cdots, s_2^*, s_5^*, \cdots\cdots \rangle$$

Then, the seventh prefix of Z^* must be $Z^*_{1..7} = \langle s_1^*, s_2^*, s_1^*, s_3^*, s_4^*, s_2^*, s_5^* \rangle$. Here, one sees that (at least) the leftmost occurrence of every s_i^* for $i = 1, 2, \cdots, c$ must be included in both the nth prefix $X^r_{1..n}$ of X^r and the mth prefix $Y^r_{1..m}$ of Y^r. As a result, we can obtain a repetition-free common subsequence of $X^r_{1..n} = X$ and $Y^r_{1..m} = Y$ of length at least c. For the above example, a repetition-free subsequence $\langle s_1^*, s_2^*, s_3^*, s_4^*, s_5^* \rangle$ of length $c = 5$ can be obtained from X and Y by removing the second s_1^* and the second s_2^* from $Z^*_{1..7}$. This completes the proof. $\square$

If every symbol appears more times, then we can prove the APX-hardness of r-RBLCS by providing a *gap-preserving reduction* from RFLCS to r-RBLCS:

Theorem 7. *For a pair of sequences X and Y and any integer $r \geq 2$, r-RBLCS is APX-hard even if $occ(X, s_i) \in \{r, 2r\}$ and $occ(Y, s_i) \in \{r, 2r\}$ hold for every symbol $s_i \in S$.*

Proof. Again, suppose that a pair of sequences $X = \langle x_1, x_2, \cdots, x_n \rangle$ and $Y = \langle y_1, y_2, \cdots, y_m \rangle$ is an instance of RFLCS such that every symbol appears either once or twice in each of the two sequences. Then, we construct the following pair of sequences X^r and Y^r as an instance of r-RBLCS, which are different from the previous X^r and Y^r in the proof of Theorem 6:

$$X^r = \langle \overbrace{x_1, x_1, \cdots, x_1}^{r}, \overbrace{x_2, x_2, \cdots, x_2}^{r}, \cdots, \overbrace{x_n, x_n, \cdots, x_n}^{r} \rangle$$
$$Y^r = \langle \overbrace{y_1, y_1, \cdots, y_1}^{r}, \overbrace{y_2, y_2, \cdots, y_2}^{r}, \cdots, \overbrace{y_n, y_n, \cdots, y_m}^{r} \rangle$$

That is, the first r symbols in X^r (Y^r, resp.) are r duplicates of x_1 (y_1, resp.), the next r symbols in X^r (Y^r, resp.) are r duplicates of x_2 (y_2, resp.), etc. This completes the reduction, which can be clearly done in polynomial time.

One sees that every symbol appears exactly r or $2r$ times in each of the two sequences X^r and Y^r.

Let Z and Z^r be optimal solutions of RFLCS and r-RBLCS for the pairs (X, Y) and (X^r, Y^r), respectively. Also, let $\Gamma(n, m)$ be a parameter function of the instance pair (X, Y) such that $\Gamma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Next, we show that the above reduction satisfies the two conditions of gap-preserving reductions: (i) If $|Z| \geq \Gamma(n, m)$, then $|Z^r| \geq r \times \Gamma(n, m)$, and (ii) if $|Z| < (1 - \varepsilon)\Gamma(n, m)$ for a fixed small positive constant $\varepsilon > 0$, then $|Z^r| < r \times (1 - \varepsilon)\Gamma(n, m)$. In the following let $Z = \langle z_1^*, z_2^*, \cdots, z_c^* \rangle$ be an optimal solution for RFLCS when its instance is the pair of sequences X and Y.

(i) Suppose that $|Z| = c$, i.e., $c \geq \Gamma(n, m)$ holds. Then, we consider the following sequence Z' when its instance is the pair of sequences X^r and Y^r:

$$Z' = \langle \overbrace{z_1^*, z_1^*, \cdots, z_1^*}^{r}, \overbrace{z_2^*, z_2^*, \cdots, z_2^*}^{r}, \cdots, \overbrace{z_c^*, z_c^*, \cdots, z_c^*}^{r} \rangle$$

From the above reduction, it is clear that Z' is a common subsequence of X^r and Y^r such that each symbol z_i^* appears at most r times and thus the length of Z' is $r \times c$. Hence, $|Z^r| \geq r \times c = r \times \Gamma(n, m)$ holds.

(ii) Suppose that the length $|Z| = c$ of the optimal solution Z of RFLCS is less than $(1 - \varepsilon)\Gamma(n, m)$. Also, suppose for the purpose of obtaining a contradiction that Z^r consists of at least $c + 1$ different symbols, say, z_1^* through z_{c+1}^*. For example, suppose that $c = 8$ and Z^r consists of nine symbols z_1^* through z_9^* as follows:

$$Z^r = \langle z_1^*, z_2^*, z_3^*, z_1^*, z_4^*, z_2^*, z_5^*, z_4^*, z_6^*, z_7^*, z_8^*, z_7^*, z_9^* \rangle$$

We can assume that the leftmost occurrence of each z_i^* appears in the subscript order in Z^r, i.e., the first symbol is z_1^*, the second symbol is z_2^*, etc, as shown above. Then, one can verify that the above sequence Z^r includes $\langle z_1^*, z_2^*, z_3^*, z_4^*, z_5^*, z_6^*, z_7^*, z_8^*, z_9^* \rangle$ as a repetition-free subsequence. More generally, the sequence $Z'' = \langle z_1^*, z_2^*, \cdots, z_{c+1}^* \rangle$ of length $c + 1$ must be a repetition-free common subsequence of X and Y, which is a contradiction.

If the optimal solution Z^r of r-RBLCS has at most c different symbols, then the length of Z^r is at most $r \times c$, which is less than $r \times (1-\varepsilon)\Gamma(n,m)$. This completes the proof. $\square$

6. Conclusion

We have studied a new variant of the Longest Common Subsequence problem, called the Repetition-Bounded Longest Common Subsequence problem (RBLCS), and its special problem, called the r-Repetition-Bounded Longest Common Subsequence problem (r-RBLCS). For $r = 1$, 1-RBLCS is known as the Repetition-Free Longest Common Subsequence problem. We first showed that for 1-RBLCS there is a simple exact algorithm whose running time is $O(1.44225^n)$. Then, for RBLCS, we designed a DP-based exact algorithm whose running time is $O(1.44225^n)$. In particular, the DP-based algorithm can solve 1-RBLCS in $O(1.41422^n)$ time. To see that reducing the time complexity from $O(1.44225^n)$ to $O(1.41422^n)$ can be of practical importance, consider for example the case of $n = 100$ and observe that 1.41422^{100} is seven times smaller than 1.44225^{100}. Hence, a promising direction for future research is to design faster exact exponential-time algorithms for RBLCS. Another challenge is to develop efficient approximation algorithms for RBLCS.

Acknowledgments. This work was partially supported by PolyU Fund 1-ZE8L, the Natural Sciences and Engineering Research Council of Canada, JST CREST JPMJR1402, and Grants-in-Aid for Scientific Research of Japan (KAKENHI) Grant Numbers JP17K00016, JP17K00024, JP17K19960 and JP17H01698.

References

[1] Adi, S.S., Braga, M.D.V., Fernandes, C.G., Ferreira, C.E., Martinez, F.V., Sagot, M.-F., Stefanes, M.A., Tjandraatmadja, C., Wakabayashi, Y.: Repetition-free longest common subsequence. *Disc. Appl. Math.*, 158, pp. 1315–1324 (2010)

[2] Aho, A., Hopcroft, J., Ullman, J.: *Data Structures and Algorithms.* Addison-Wesley (1983)

[3] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. *J. Molecular Biology*, 215(3), pp. 403–410 (1990)

[4] Beal, R., Afrin, T., Farheen, A., Adjeroh, D.: A new algorithm for “the LCS problem” with application in compressing genome resequencing data. *Proc. BIBM*, pp. 69–74 (2015)

[5] Bergroth, L., Hakonen, H., Raita, T.: A survey of longest common subsequence algorithms. *Proc. SPIRE*, pp. 39–48 (2000)

[6] Blin, G., Bonizzoni, P., Dondi, R., Sikora, F.: On the parameterized complexity of the repetition free longest common subsequence problem. *Info. Proc. Lett.*, 112(7), pp. 272–276 (2012)

[7] Blum, C., Blesa, M.J., Calvo, B.: Beam-ACO for the repetition-free longest common subsequence problem. *Proc. EA 2013*, pp. 79–90 (2014)

[8] Blum, C., Blesa, M.J.: Construct, merge, solve and adapt: application to the repetition-free longest common subsequence problem. *Proc. EvoCOP2016*, pp. 46–57 (2016)

[9] Blum, C., Blesa, M.J.: A comprehensive comparison of metaheuristics for the repetition-free longest common subsequence problem. *J. Heuristics*, 24(3), pp. 551–579 (2018)

[10] Bonizzoni, P., Della Vedova, G., Dondi, R., Fertin, G., Rizzi, R., Vialette, S.: Exemplar longest common subsequence. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 4(4), pp. 535–543 (2007)

[11] Bonizzoni, P., Della Vedova, G., Dondi, R., Pirola, Y.: Variants of constrained longest common subsequence. *Info. Proc. Lett.*, 110(20), pp. 877–881 (2010)

[12] Bulteau, L., Hüffner, F., Komusiewicz, C., Niedermeier, R.: Multivariate Algorithmics for NP-Hard String Problems: The Algorithmics Column by Gerhard J Woeginger. Bulletin of EATCS, No.114 (2014)

[13] Castelli, M., Beretta, S., Vanneschi, L.: A hybrid genetic algorithm for the repetition free longest common subsequence problem. *Oper. Res. Lett.*, 41(6), pp. 644–649 (2013)

[14] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 3rd Ed. The MIT Press (2009)

[15] Fernandes, C.G., Kiwi, M.: Repetition-free longest common subsequence of random sequences. *Disc. Appl. Math.*, 210, pp. 75–87 (2016)

[16] Hirschberg, D.S.: Algorithms for the longest common subsequence problem, *J. ACM*, 24(4), pp. 664–675 (1977)

[17] Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, 18(6), pp. 341–343 (1975)

[18] Itoga, S.Y.: The string merging problem. *BIT*, 21(1), pp. 20–30 (1981)

[19] Maier, D.: The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2), pp. 322–336 (1978)

[20] Jiang, T., Li, M.: On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput.*, 24(5), pp. 1122–1139 (1995)

[21] Morgan, H.L.: Spelling correction in systems programs. *Comm. ACM*, 13(2), pp. 90–94 (1970)

[22] Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Molecular Biology*, 48(3), pp. 443–453 (1970)

[23] Sankoff, D.: Matching sequences under deletion/insertion constraints. *Proc. National Academy of Science USA*, 69(1), pp. 4–6 (1972)

[24] Sankoff, D.: Genome rearrangement with gene families. *Bioinformatics*, 15(11), pp. 909–917 (1999)

[25] Mincu, R.S., Popa, A.: Better heuristic algorithms for the repetition free LCS and other variants. *Proc. SPIRE*, pp. 297–310 (2018)

[26] Storer, J.A.: Data compression: methods and theory. *Computer Science Press* (1988)

[27] Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. *J. ACM*, 21(1), pp. 168–173 (1974)

Appendix: Summary of notation

Note: Some symbols that appear only in a restricted context are not listed.

S	an alphabet, i.e., a finite set of symbols
X, Y	two input sequences of symbols
Z	a common subsequence of the given two sequences
n	the length $\|X\|$ of X
m	the length $\|Y\|$ of Y
$X_{1..i}$	the ith prefix of a sequence X
$X_{j..n}$	the jth suffix of a sequence X
C	a sequence constraint, i.e., a set of sequences over an alphabet S
$C_{occ}(s)$	an occurrence constraint, i.e., a function assigning an upper bound on the number of occurrences of each symbol $s \in S$
$occ(X, s)$	the number of occurrences of symbol $s \in S$ in X
occ_{max}	the maximum number of occurrences of all the symbols $s \in S$ in X, i.e., $\mathsf{occ}_{max} = \max_{s \in S} \{occ(X, s)\}$
S_i	a set of every symbol s which appears exactly i times in X, i.e., $S_i = \{s \mid occ(X, s) = i\}$
$S_{>C_{occ}}$	a set of every symbol s which appears more than $C_{occ}(s)$ times in X, i.e., $S_{>C_{occ}} = \{s \mid occ(X, s) > C_{occ}(s)\}$
e	Euler's number
$\mathbb{R}^+$	a set of positive reals
$\mathbb{N}$	a set of positive integers